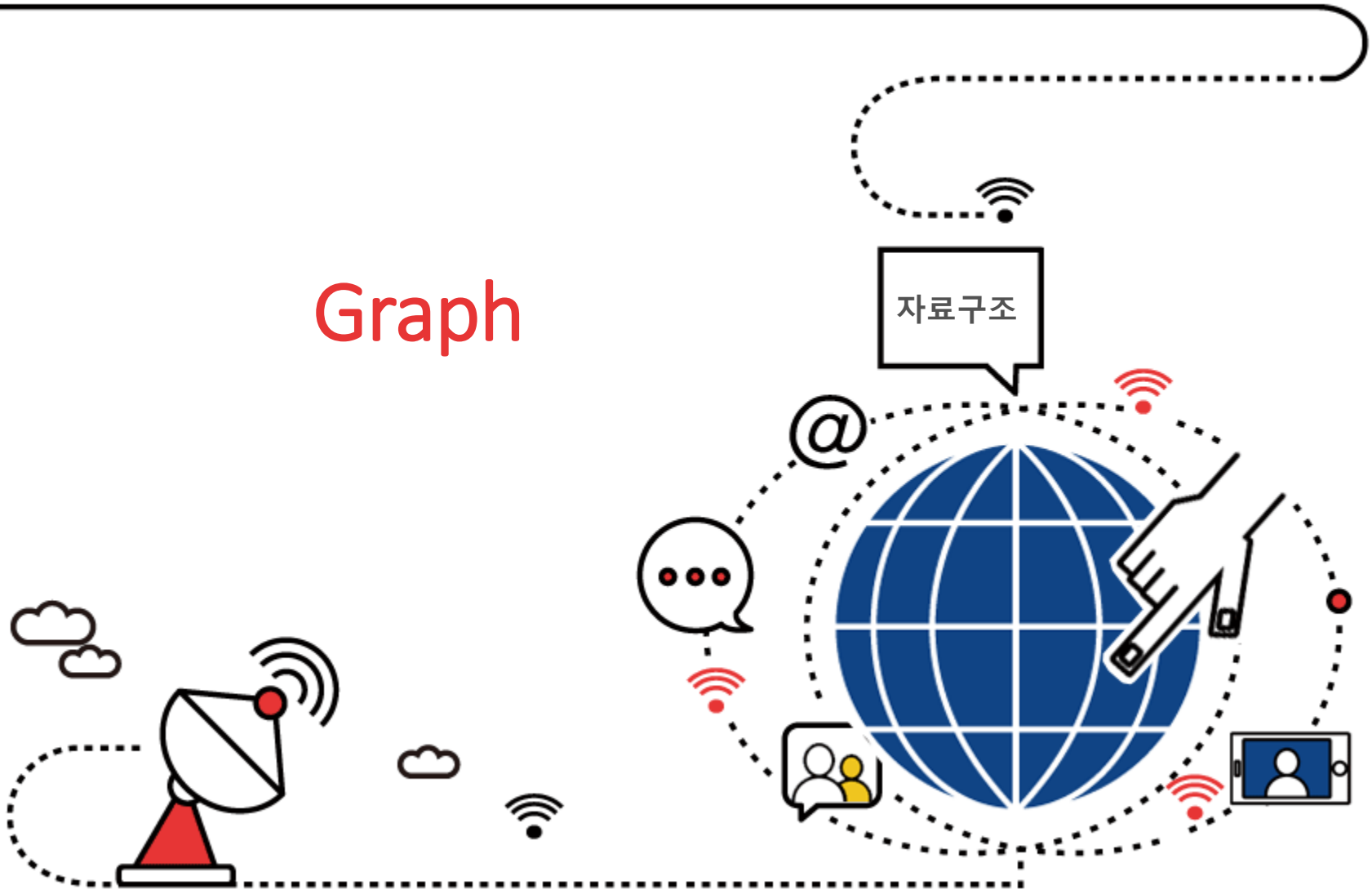
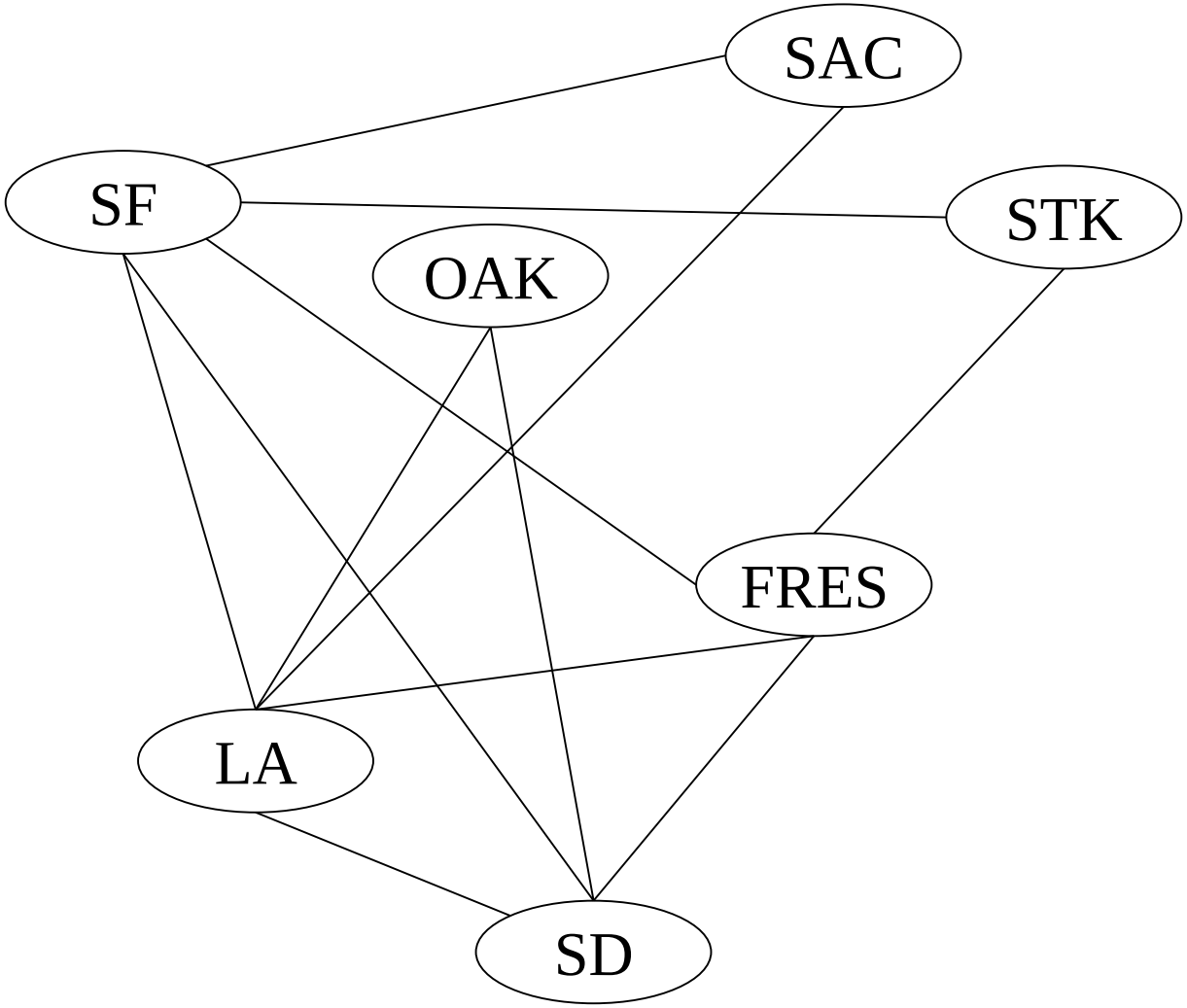
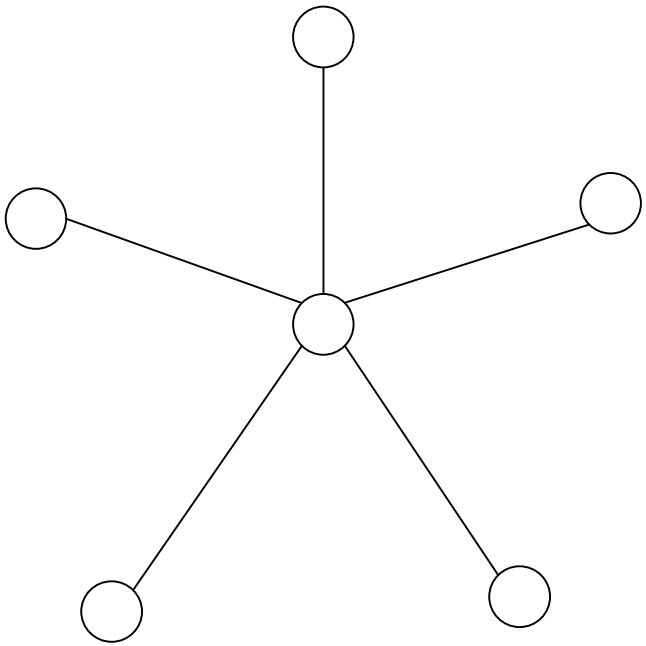
Graph



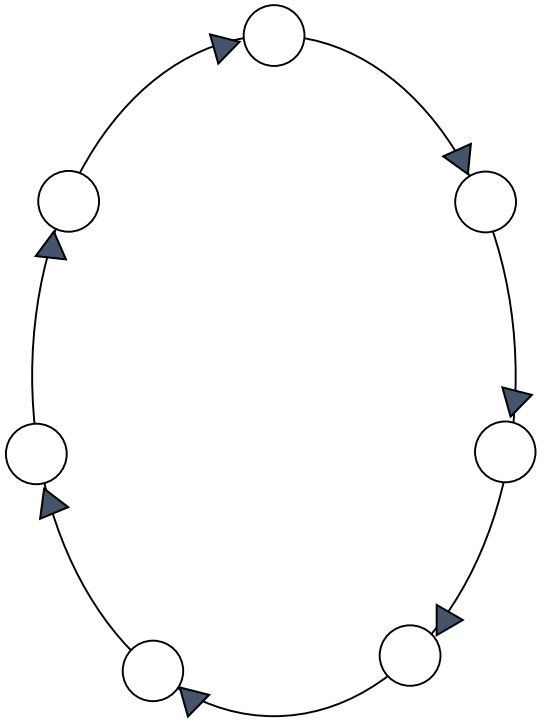
그래프 예시 - 공항



그래프 예시 - 네트워크



(a) A star network



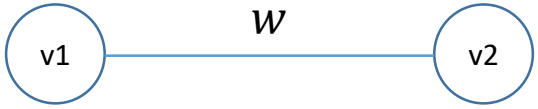
(b) A ring network

Graph 란?

- 정점(Vertex)과 간선(Edge)으로 이루어진 자료구조
- 간선의 종류에 따라 그래프 종류가 달라진다
  - Undirected Graph : 간선이 방향 X

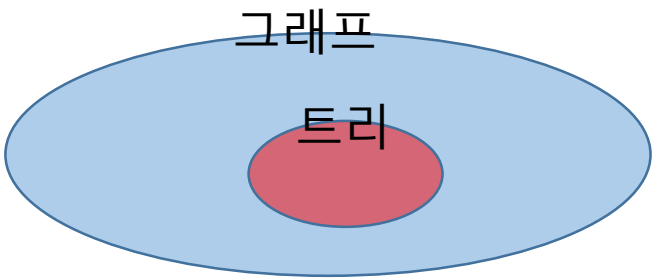
  - Directed Graph : 간선이 방향을 가리킴

  - Weighted Graph : 간선에 가중치가 있음



Graph vs Tree

그래프	트리
노드와 노드를 연결하는 간선들의 집합	Cycle이 없는 Connected Graph
Cycle 존재 가능	Cycle 존재 불가능
두 정점 사이에 여러 개의 경로 존재 가능	두 정점 사이에 반드시 1개의 경로 존재
간선의 수는 그래프에 따라 다름	간선의 수 = 정점의 수 -1



그래프 ⊃ 트리

Graph

- **Graph** $G = (V, E), n = |V|, m = |E|$
- **Weighted Graph** $G = (V, E, W)$
- **Graph의 표현방법(Representation)은 2가지**
  - **Adjacency Matrix Representation(인접 행렬)**
  - **Adjacency List Representation (인접 리스트)**

▪ no parallel edges ▪ no self-loops	Adjacency List	Adjacency Matrix
Space	$n + m$	n^2
v.incidentEdges()	deg(*v*)	*n*
v.isAdjacentTo (*w*)	min(deg(*v*), deg(*w*))	1
insertVertex(*o*)	1	n^2
insertEdge(*v*, *w*, *o*)	1	1
eraseVertex(*v*)	deg(*v*)	n^2
eraseEdge(*e*)	1	1

표현법의 차이점

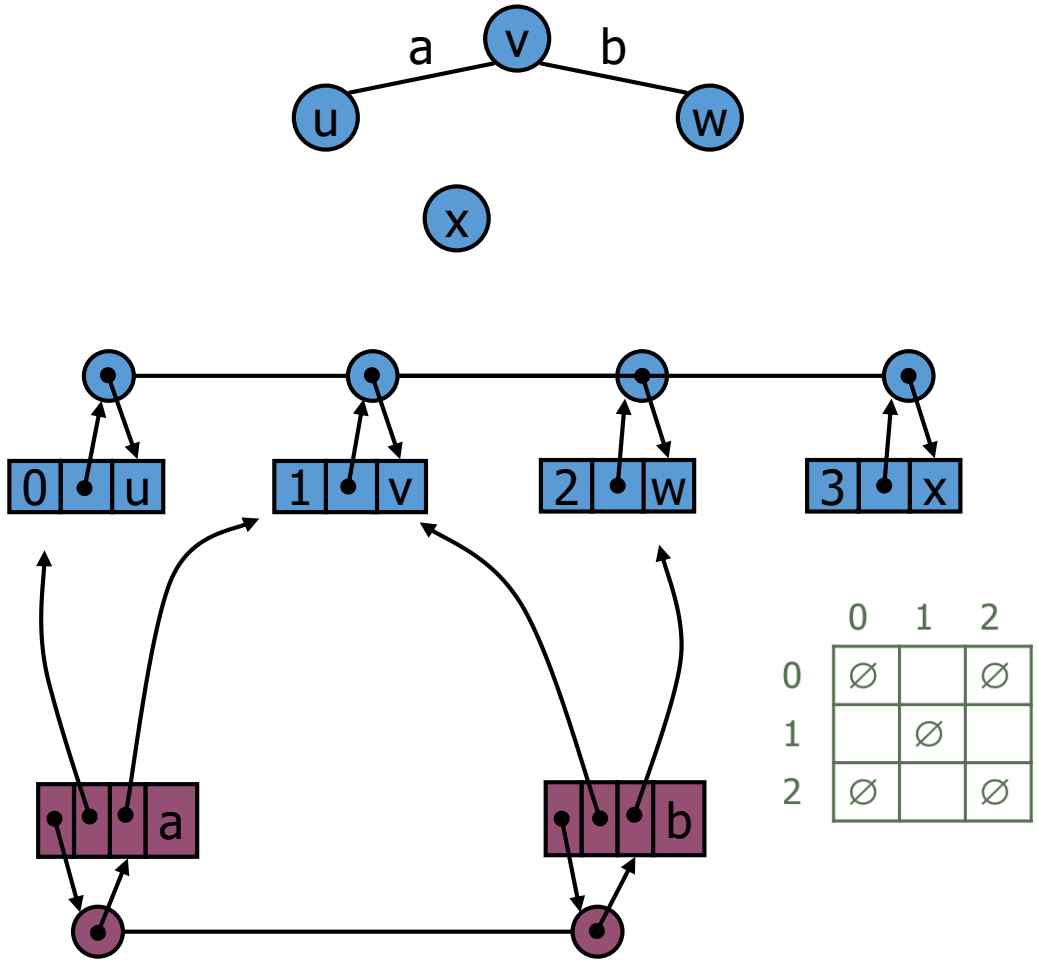
인접 행렬

- 노드 i와 j의 인접(adjacent) 여부를 $O(1)$ 시간에 확인 가능
- 밀집한(dense) 그래프에서 효율적
- 구현이 쉽다

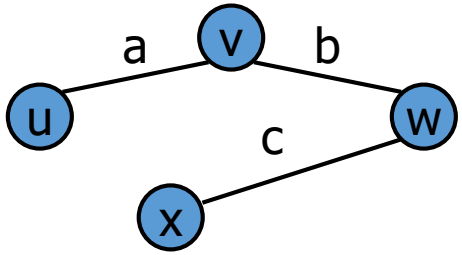
인접 리스트

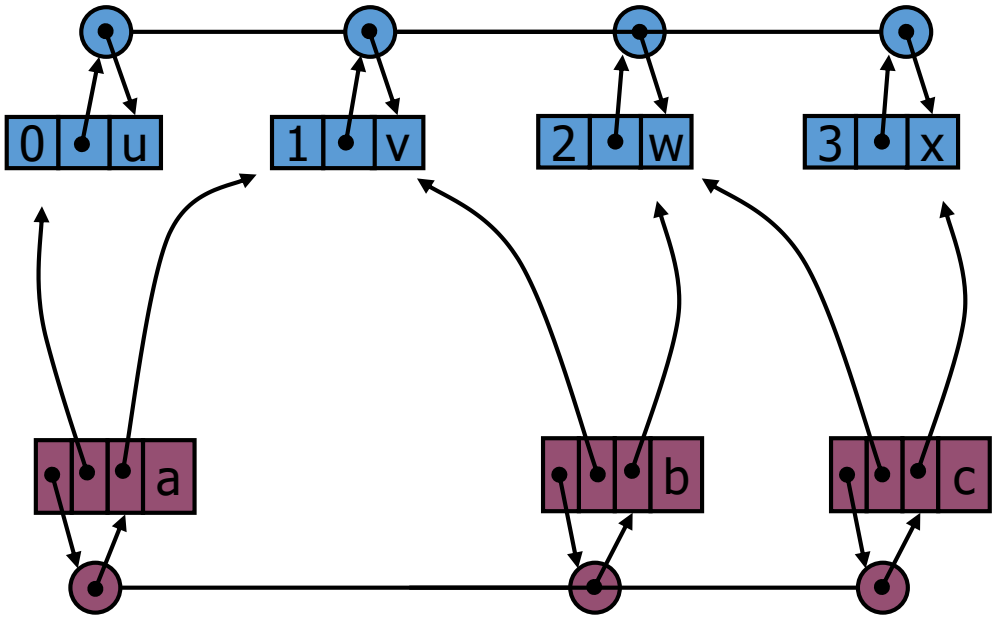
- 노드 v에 인접한(incident) 간선들의 정보를 $O(degree(v))$ 시간에 확인 가능
- 드문(sparse) 그래프에서 효율적
- 구현이 상대적으로 복잡하다

정점 삽입



간선 삽입





	0	1	2
0	∅		∅
1		∅	
2	∅		∅

	0	1	2	3
0	∅		∅	∅
1		∅		∅
2	∅		∅	
3	∅	∅		∅

코드

```cpp
#include<iostream>
#include<vector>

#define M [redacted]

using namespace std;

class vertex {
private:
    int degree;
public:
    vertex() { this->degree = 0; }
    void increase_deg() {
        this->degree++;
    }
    void decrease_deg() {
        this->degree--;
    }
};

class edge {
private:
    vertex* src;
    vertex* dst;
public:
    edge(vertex* src, vertex* dst) {
        this->src = src;
        this->dst = dst;
    }
};
```

코드

```
class graph {
private:
    vertex** v;
    edge*** matrix;
    vector<edge*> e;
    int v_sz;
    int max_sz;
public:
    graph(int sz) {
        this->max_sz = sz;
        this->v_sz = 0;

        this->v = new vertex*[sz+1];
        for (int i = 0; i <= sz; i++) v[i] = NULL;

        this->matrix = new edge**[sz+1];
        for (int i = 0; i <= sz; i++) {
            matrix[i] = new edge*[sz+1];
            for (int j = 0; j <= sz; j++) matrix[i][j] = NULL;
        }
    }
    void insert_vertex(int n) {
        if (v_sz + 1 > max_sz) return; // graph에 더이상 정점을 삽입 할수 없을 경우 바로 종료
        if (v[n] == NULL) {
            vertex* new_v = new vertex();
            v[n] = new_v;
            this->v_sz++;
        }
        else {
            return; // 이미 정점이 있다면, 바로 종료
        }
    }
```

코드

```
    void insert_edge(int src, int dst) {

        if (v[src] == NULL || v[dst] == NULL) return; // 삽입될 edge의 양끝 vertex중 하나라도 없으면, 바로 종료
        if (matrix[src][dst] != NULL || matrix[dst][src] != NULL) {
            cout << 0 << endl;
            return; // 이미 간선이 존재 하기 때문에, 바로 종료
        }
        edge* new_e = new edge(v[src], v[dst]);
        matrix[src][dst] = new_e;
        matrix[dst][src] = new_e; // 무향 그래프이므로, 양쪽에 다 포인터설정

        v[src]->increase_deg();
        v[dst]->increase_deg();

        e.push_back(new_e);
    }

    void erase_edge(int src, int dst) {
        if (matrix[src][dst] == NULL || matrix[dst][src] == NULL) return; // 간선이 존재하지 않는다면, 바로 종료
        v[src]->decrease_deg();
        v[dst]->decrease_deg();

        for (int i = 0; i < e.size(); i++) {
            if (e[i] == matrix[src][dst] || e[i] == matrix[dst][src]) {
                e.erase(e.begin() + i);
                break;
            }
        }

        matrix[src][dst] = NULL;
        matrix[dst][src] = NULL; // 무향 그래프이므로, 양쪽에 다 포인터설정
    }
```


Thank you